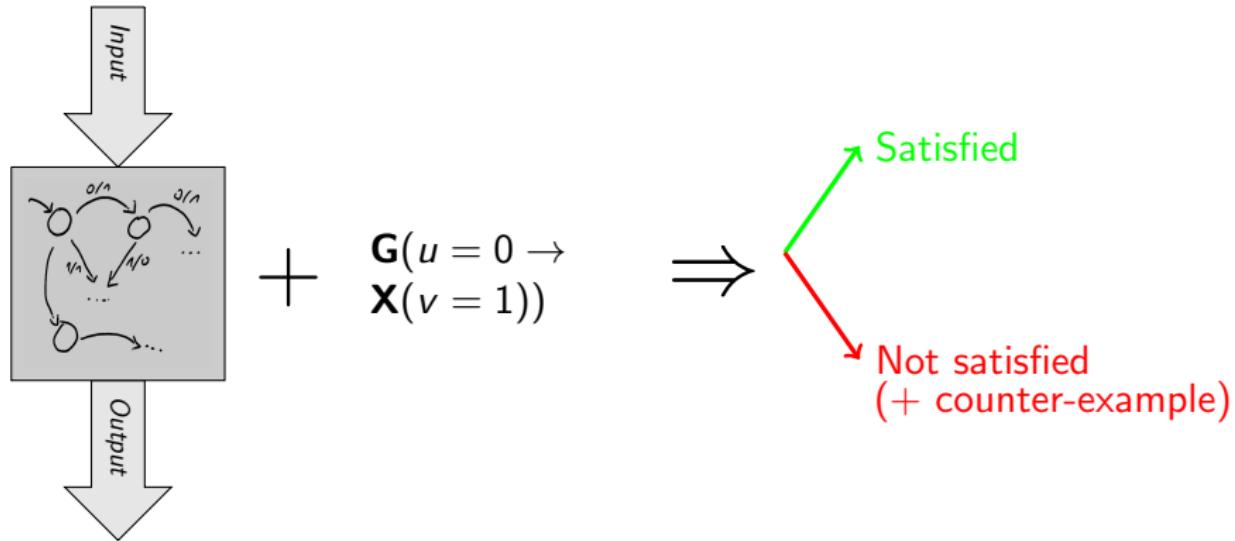


Symbolically Synthesizing Small Circuits

Rüdiger Ehlers¹ Robert Könighofer² Georg Hofferek²

FMCAD 2012 – 24th October 2012

Verification:

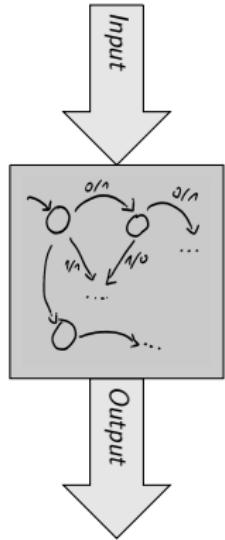
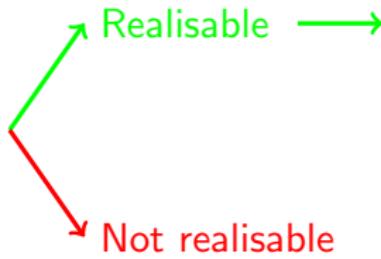


Synthesis:

$$\mathbf{G}(u = 0 \rightarrow \\ \mathbf{X}(v = 1))$$

+

$$\text{Input} = \{u, \dots\} \\ \text{Output} = \{v, \dots\}$$



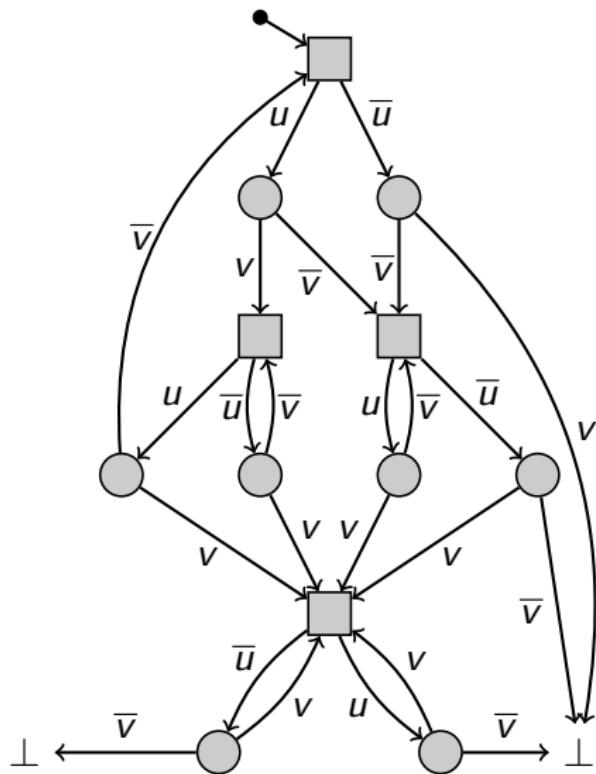
Synthesis in a nutshell

$$\begin{aligned}\mathbf{G}(u = 0) \\ \rightarrow \mathbf{X}(v = 1))\end{aligned}$$

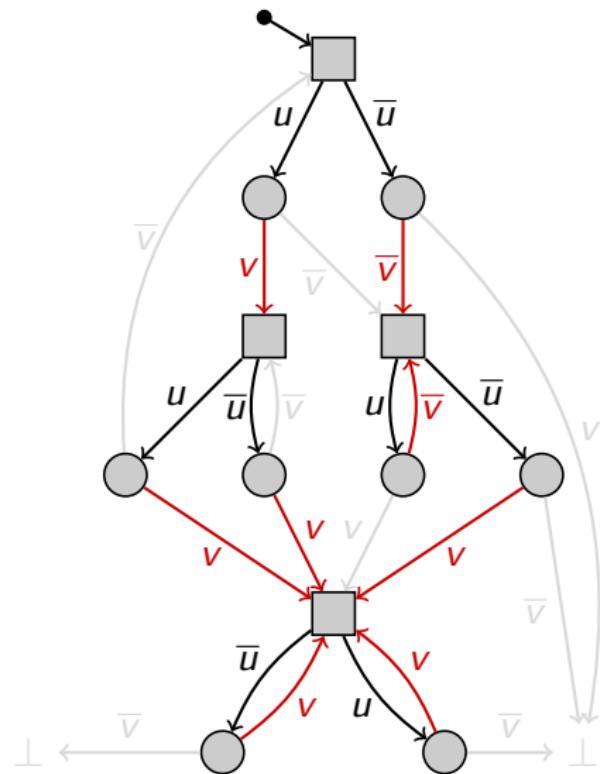
+

$$\begin{aligned}\text{Input} &= \{u, \dots\} \\ \text{Output} &= \{v, \dots\}\end{aligned}$$

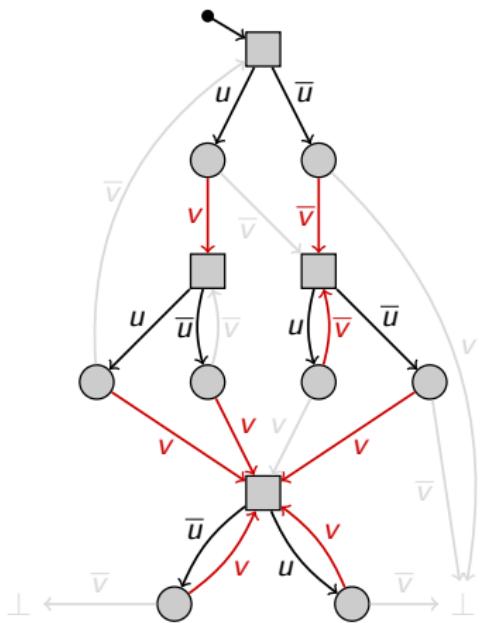
Synthesis in a nutshell



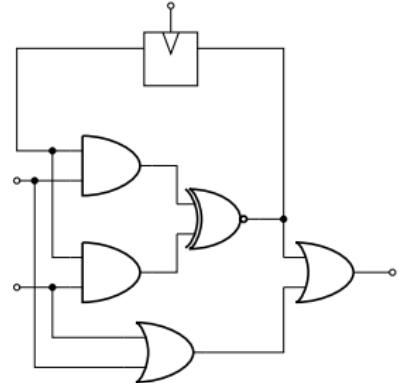
Synthesis in a nutshell



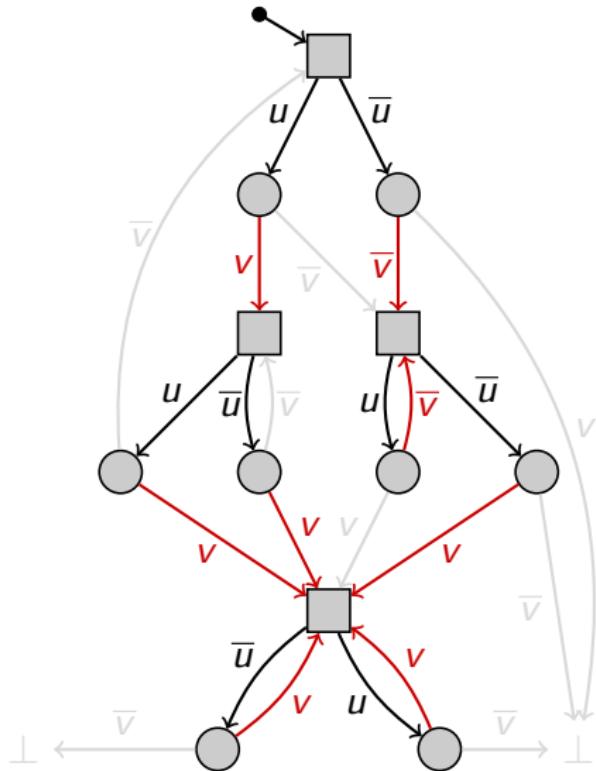
Synthesis in a nutshell



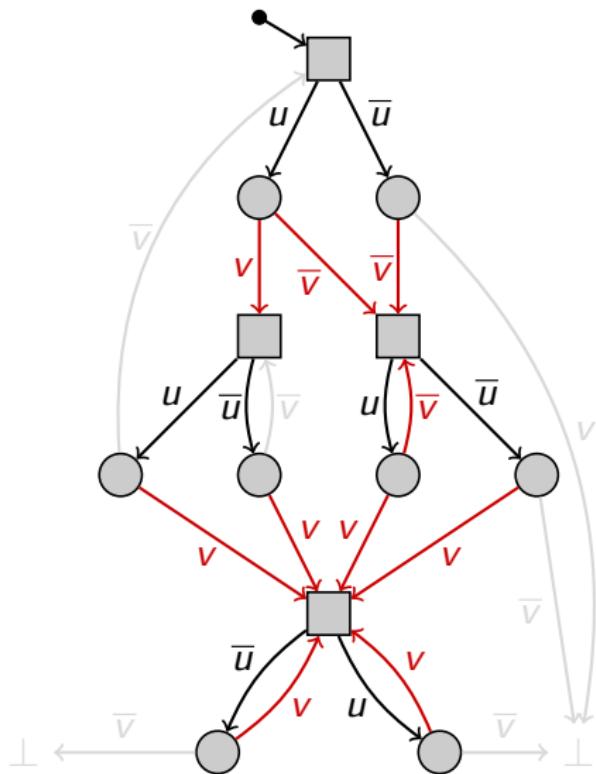
```
int main() {
    bool s = 1;
    while (true) {
        bool b = in-u();
        out-v(b | s);
        s = s ⊕ u;
        wait();
    }
}
```



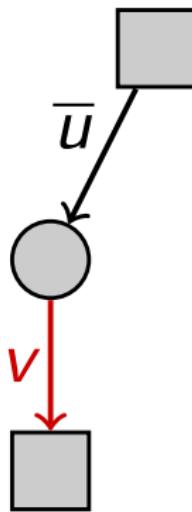
On general strategies



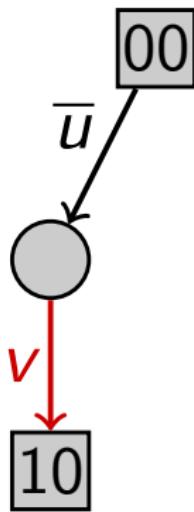
On general strategies



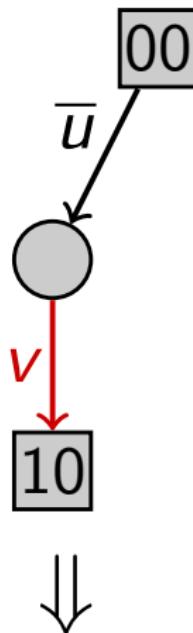
Encoding general strategies



Encoding general strategies

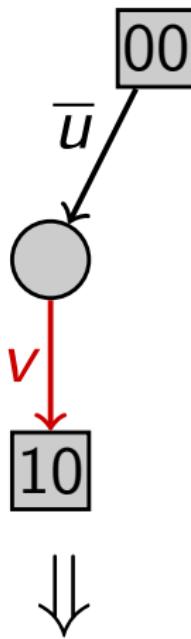


Encoding general strategies

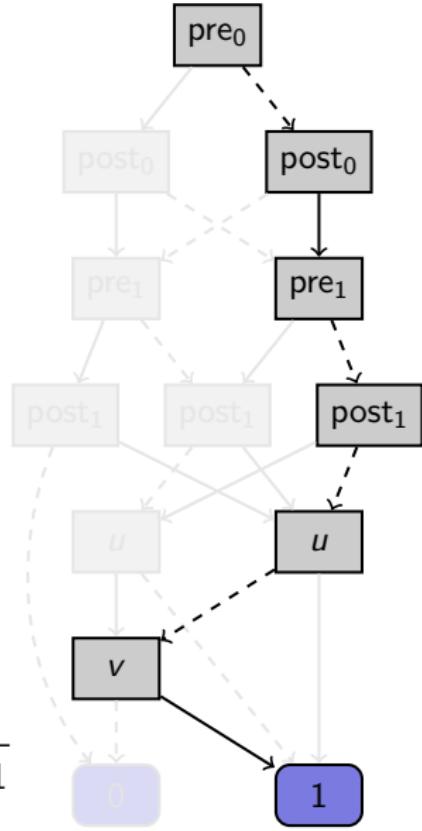


$$\overline{pre_0} \wedge \overline{pre_1} \wedge \overline{u} \wedge v \wedge pre_0 \wedge \overline{pre_1}$$

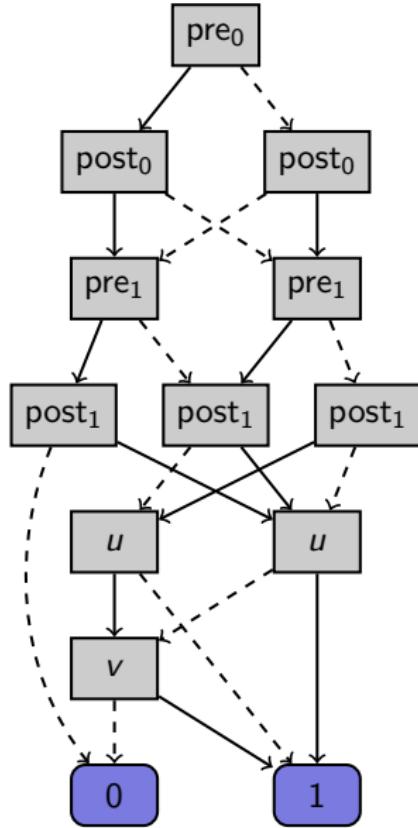
Encoding general strategies



$$\overline{\text{pre}_0} \wedge \overline{\text{pre}_1} \wedge \bar{u} \wedge v \wedge \text{pre}_0 \wedge \overline{\text{pre}_1}$$



Building circuits from general strategies



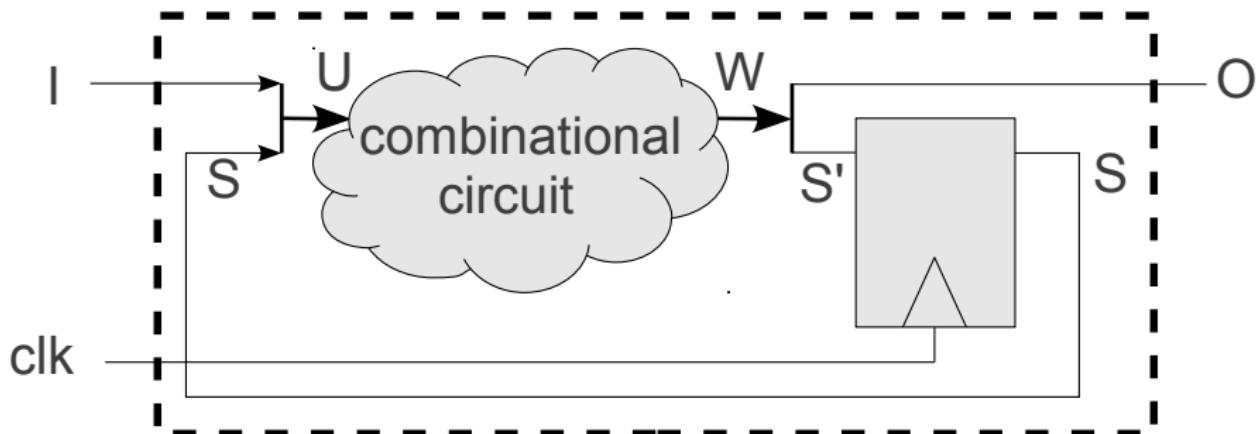
General strategies ...

- ... represent allowed behaviour of the system to synthesize
- ... are typically given symbolically
- ... are often **huge**. To have 64 state bits in the BDD is common.

What is a suitable implementation?

Any implementation that is a specialisation of the general strategy is fine.

Building circuits from general strategies

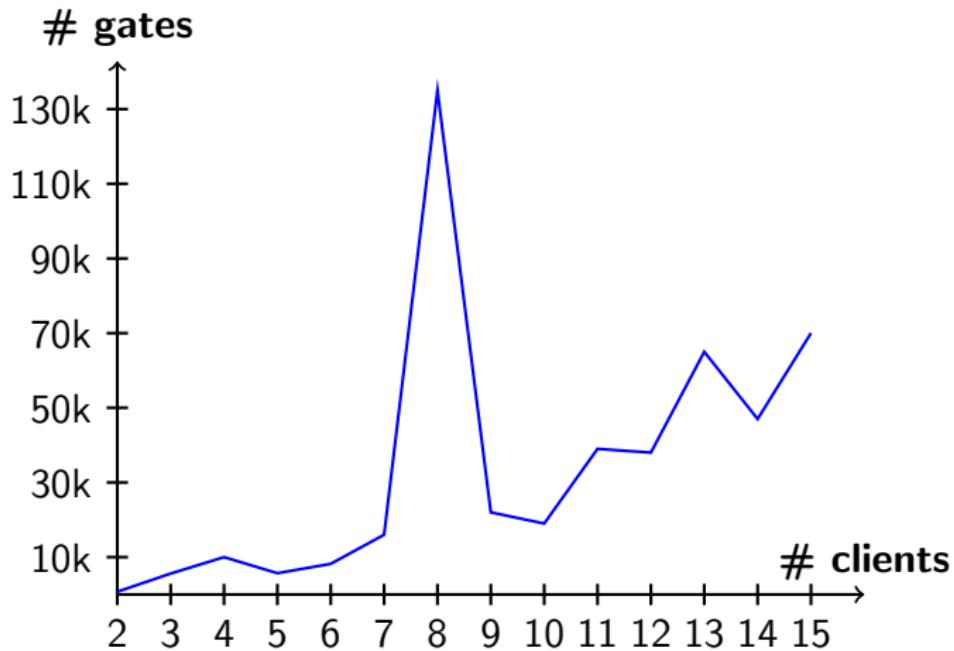


Previous approaches to building the combinational circuit

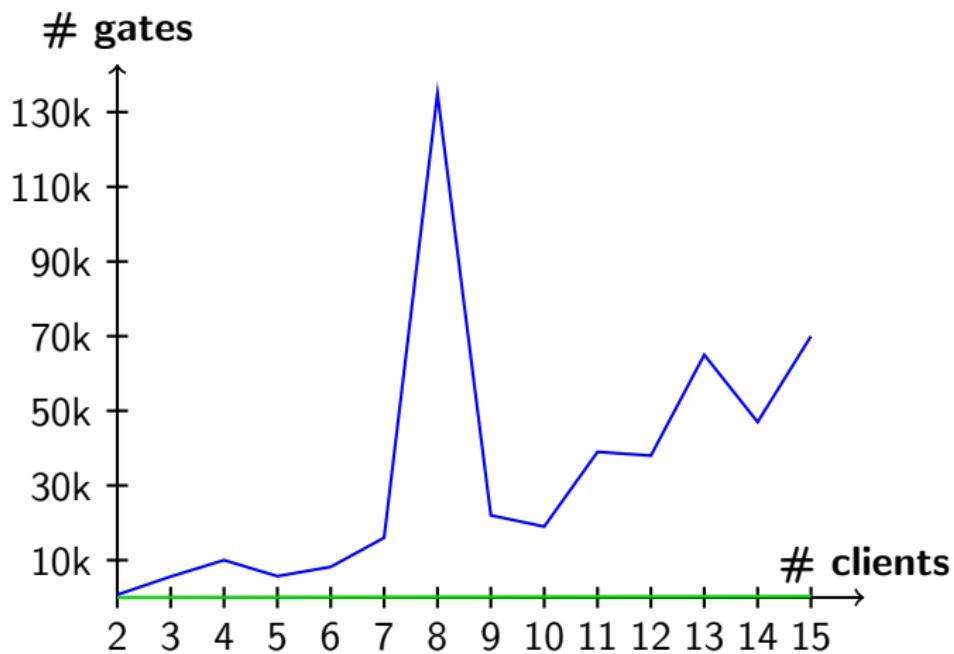
Experiences in the synthesis domain

- Kukula and Shipley (2000) - huge, deep circuits
- Bañeres et al. (2004) - extremely slow on synthesis problems
- Bloem et al. (2007b) - best approach so far, circuits are still large
- Jiang et al. (2009) - generates larger circuits than the approach by Bloem et al. (2007b)
- ...

Example: AMBA AHB bus arbiter (Bloem et al., 2007b)



Example: AMBA AHB bus arbiter (Bloem et al., 2007b)



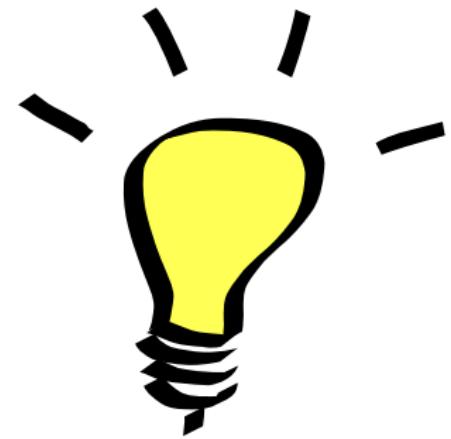
Our new approach

Main idea

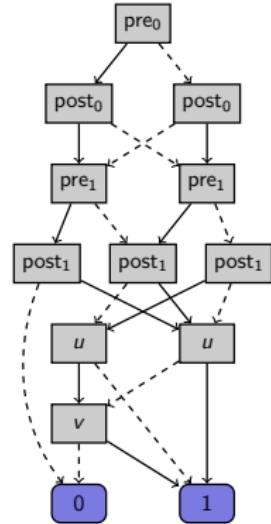
Use **computational learning of Boolean functions** as main tool for computing small circuit implementations.

Benefits

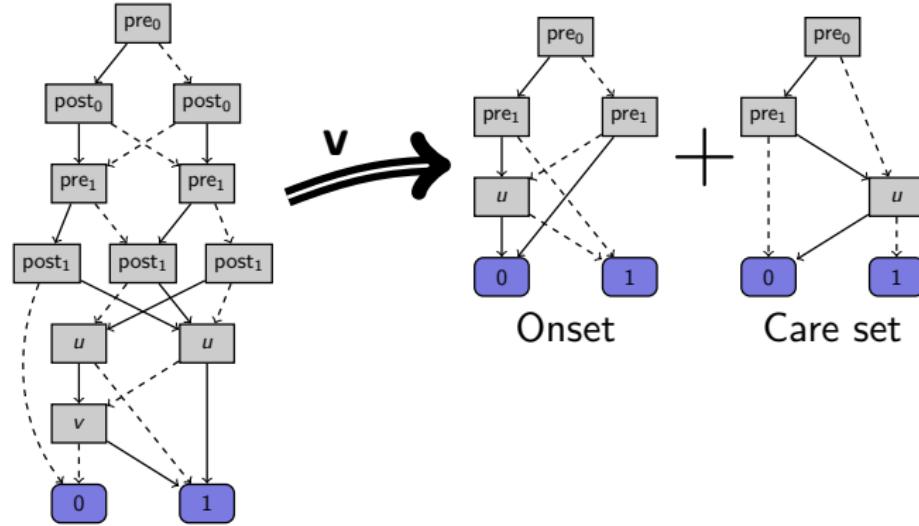
- We can efficiently benefit from a large number of *don't care* cases
- We obtain *shallow* circuits



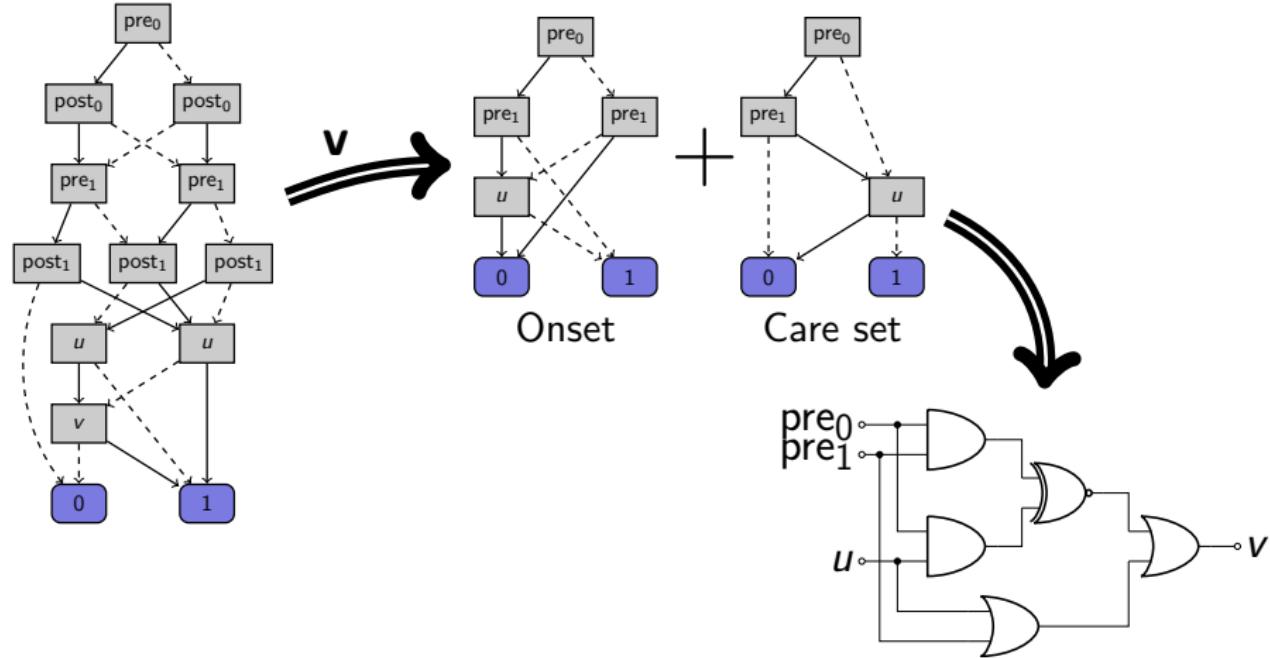
Bit-by-bit circuit extraction approach



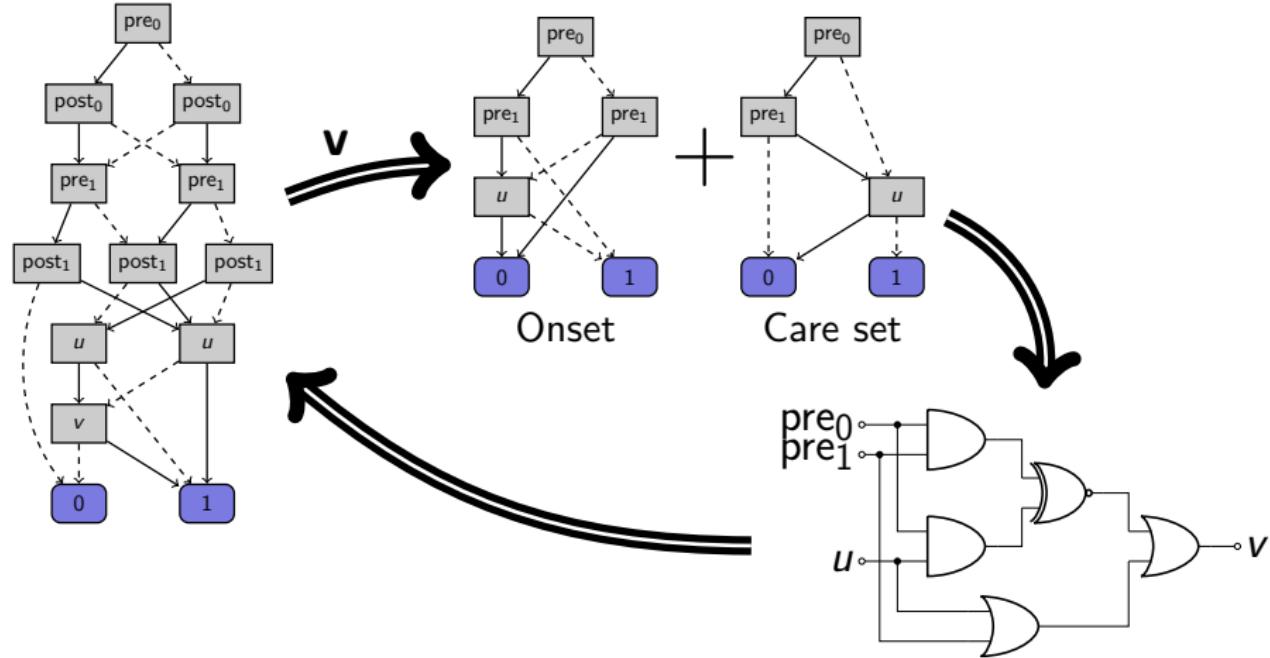
Bit-by-bit circuit extraction approach



Bit-by-bit circuit extraction approach

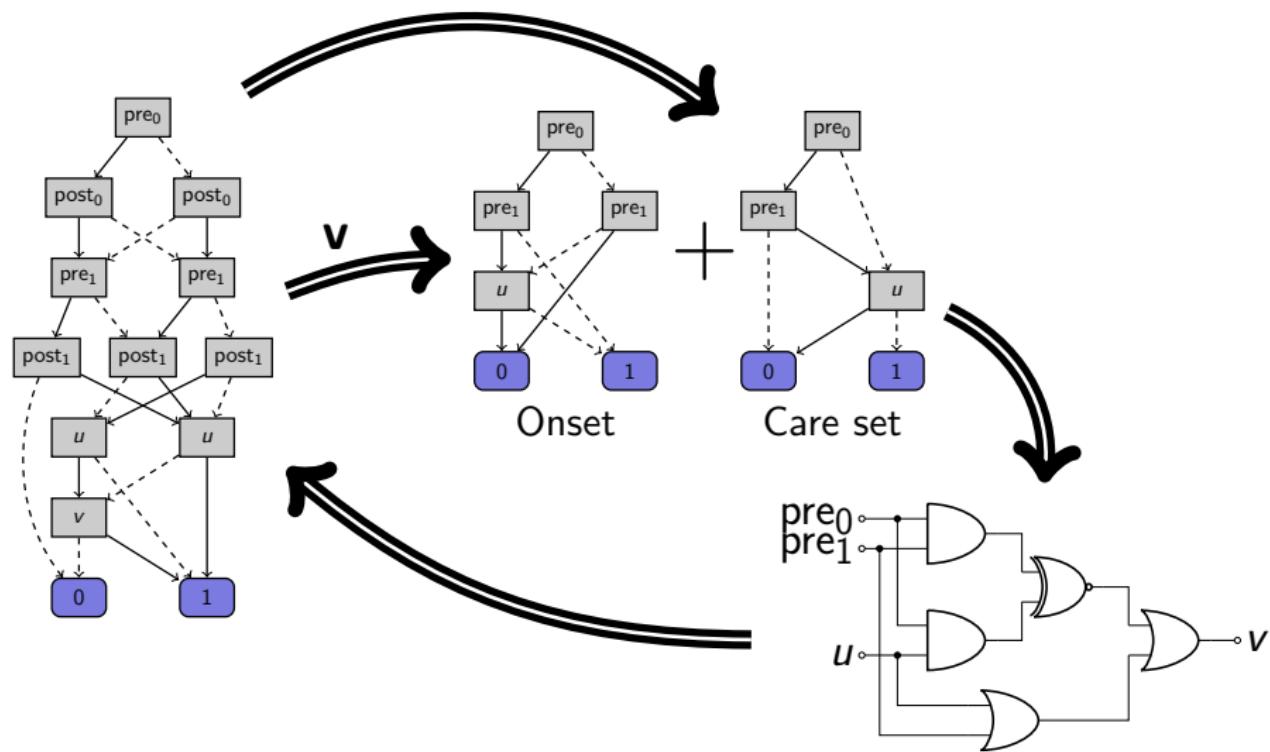


Bit-by-bit circuit extraction approach

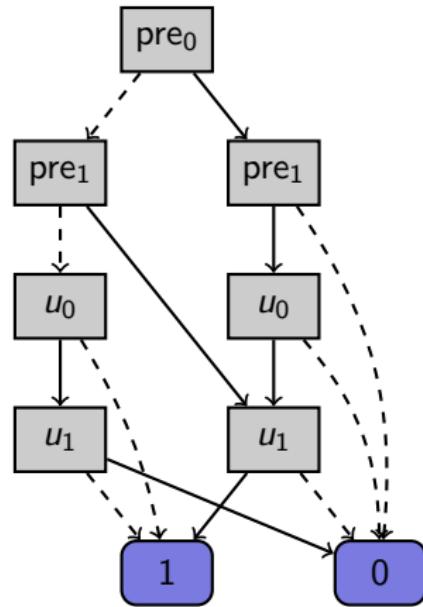


Bit-by-bit circuit extraction approach

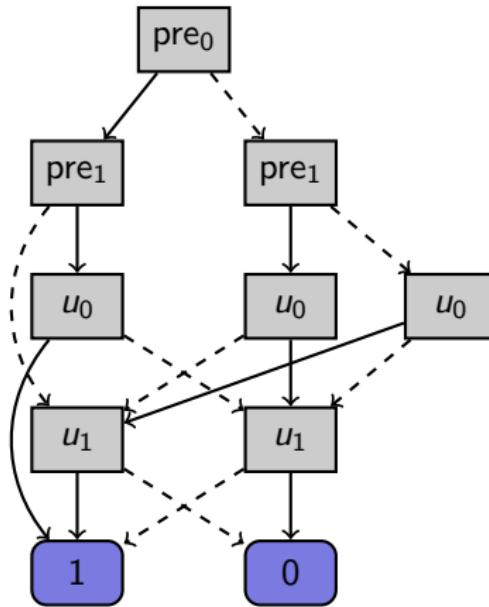
Reachable states/positions



Boolean formula learning (DNF)

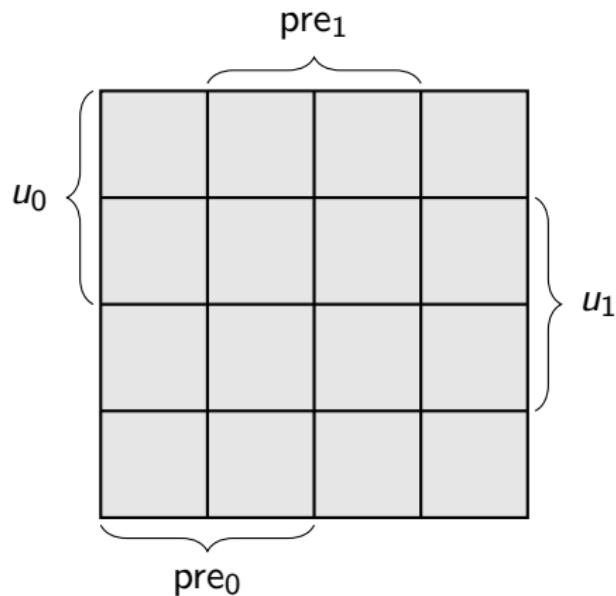


Onset



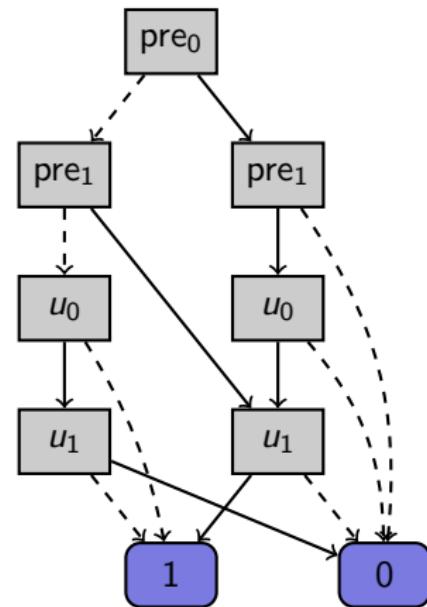
Care Set

Boolean formula learning (DNF)



Boolean formula learning (DNF)

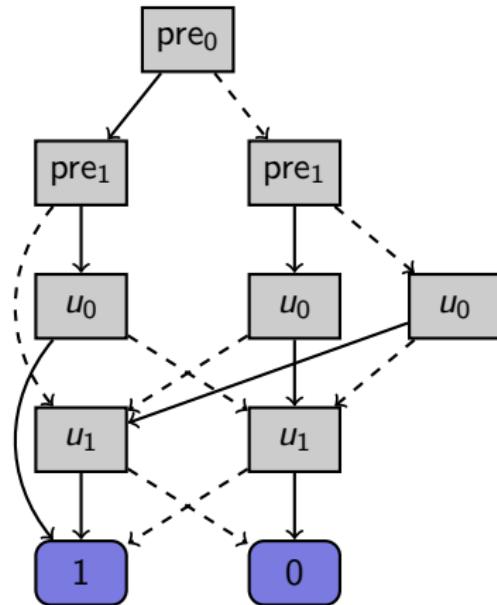
		pre ₁	
		0	1
u ₀	0	0	0
	1	1	0
u ₁	0	0	1
	1	0	1



Onset

Boolean formula learning (DNF)

				pre_1
				u_0
				u_1
X	0	0	X	
0	1	X	0	
0	X	1	X	
X	0	X	1	



Care Set

Boolean formula learning (DNF)

pre ₁				
u ₀ {	X	0	0	X
	0	1	X	0
	0	X	1	X
	X	0	X	1
pre ₀				
} u ₁				

Boolean formula learning (DNF)

pre ₁			
u_0	X	0	0
	0	1	X
	0	X	1
	X	0	X
pre ₀			

Candidate solution:

Boolean formula learning (DNF)

pre ₁			
u_0	X	0	0
	0	1	X
	0	X	1
	X	0	X
pre ₀			

Candidate solution:

$$\psi = \text{false}$$

Boolean formula learning (DNF)

pre ₁			
u ₀	0	0	X
0	1	X	0
0	X	1	X
X	0	X	1

pre₀

Candidate solution:

$$\psi = \text{false}$$

Boolean formula learning (DNF)

pre ₁			
u_0	X	0	0
	0	1	X
	0	X	1
	X	0	X
pre ₀			

Candidate solution:

$$\psi = \text{false}$$

Boolean formula learning (DNF)

pre ₁			
u ₀	0	0	X
0	1	X	0
0	X	1	X
X	0	X	1

pre₀

Candidate solution:

$$\psi = \text{false}$$

Boolean formula learning (DNF)

pre ₁			
u ₀	0	0	X
0	1	X	0
0	X	1	X
X	0	X	1

pre₀

Candidate solution:

$$\psi = \text{false}$$

Boolean formula learning (DNF)

pre ₁			
u_0	X	0	0
	0	1	X
	0	X	1
	X	0	X

pre₀

Candidate solution:

$$\psi = \text{false}$$

Boolean formula learning (DNF)

pre ₁			
u_0	X	0	0
	0	1	X
	0	X	1
	X	0	X
pre ₀			

Candidate solution:

$$\psi = \text{false}$$

Boolean formula learning (DNF)

pre ₁			
u_0	X	0	0
	0	1	X
	0	X	1
	X	0	X

pre₀

Candidate solution:

$$\psi = \text{false}$$

Boolean formula learning (DNF)

		pre ₁	
		0	0
u ₀	0	1	X
	0	X	1
	X	0	X
		X	1

Diagram illustrating a 4x4 matrix representing Boolean values (0, 1, X) for variables pre₀, pre₁, u₀, and u₁. The matrix is partitioned into four quadrants by curly braces. The top-right quadrant (pre₁, u₀) contains 0s and Xs. The bottom-left quadrant (pre₀, u₀) contains 0s and Xs. The bottom-right quadrant (pre₁, u₁) contains 1s and Xs. The top-left quadrant (pre₀, u₁) contains 1s and Xs. A blue rounded rectangle highlights the bottom-right quadrant.

Candidate solution:

$$\psi = \neg \text{pre}_0 \wedge \neg u_0$$

Boolean formula learning (DNF)

pre ₁			
u ₀	0	0	X
0	1	X	0
0	X	X	X
X	0	X	X

pre₀

Candidate solution:

$$\psi = \neg \text{pre}_0 \wedge \neg u_0$$

Boolean formula learning (DNF)

		pre ₁	
		0	0
u ₀	X	0	X
	0	1	X
	0	X	X
	X	0	X

pre₀

Candidate solution:

$$\psi = \neg \text{pre}_0 \wedge \neg u_0$$

Boolean formula learning (DNF)

pre ₁			
u ₀	0	0	X
0	1	X	0
0	X	X	X
X	0	X	X

Diagram illustrating a 4x4 matrix representing Boolean values. The columns are labeled pre₁ at the top and u₀ on the left. The rows are labeled pre₀ at the bottom. The matrix contains the following values:

	pre ₁	pre ₁	pre ₁
u ₀	X	0	0
pre ₀	0	1	X
	0	X	X
	X	0	X

A blue oval highlights the cell containing '1' at the intersection of u₀ and pre₁. The entire row u₀ and column pre₁ are highlighted in green.

Candidate solution:

$$\psi = \neg \text{pre}_0 \wedge \neg u_0$$

Boolean formula learning (DNF)

		pre ₁	
		0	0
u ₀	X	0	X
	0	1	X
	0	X	X
	X	0	X

Diagram illustrating a 4x4 matrix representing Boolean values (0, X, or 1). The columns are labeled pre₀ and pre₁. The rows are labeled u₀ and u₁. A value '1' is highlighted in a green circle at the intersection of row u₀ and column pre₁.

Candidate solution:

$$\psi = \neg \text{pre}_0 \wedge \neg u_0$$

Boolean formula learning (DNF)

				pre ₁
u ₀		0		0
		0	1	X
X	0	X	0	X
0	X	X	X	X
X	0	X	X	X

Diagram illustrating a candidate solution for Boolean formula learning (DNF). The matrix shows four columns and five rows of variables. The first column is labeled u_0 and the last column is labeled u_1 . The first row is labeled pre_1 and the last row is labeled pre_0 . A blue oval highlights the value 1 in the second row, third column, which corresponds to the variable u_1 .

Candidate solution:

$$\psi = \neg pre_0 \wedge \neg u_0$$

Boolean formula learning (DNF)

		pre ₁	
		0	0
u ₀	X	0	X
	0	1	X
	0	X	X
	X	0	X

Diagram illustrating a 4x4 matrix representing Boolean values (0, X, or 1). The columns are labeled pre₀ and pre₁. The rows are labeled u₀ and u₁. A value '1' is highlighted in a green circle at the intersection of row u₀ and column pre₁.

Candidate solution:

$$\psi = \neg \text{pre}_0 \wedge \neg u_0$$

Boolean formula learning (DNF)

		pre ₁	
		0	0
u ₀	X	0	X
	0	1	X
	0	X	X
	X	0	X

Diagram illustrating a 4x4 matrix representing Boolean values (0, X, or 1). The columns are labeled pre₀ and pre₁. The rows are labeled u₀ and u₁. A blue oval highlights the cell at (u₀, pre₁) containing the value 1.

Candidate solution:

$$\psi = \neg \text{pre}_0 \wedge \neg u_0$$

Boolean formula learning (DNF)

		pre ₁	
		0	0
u ₀	X	0	X
	0	1	X
	0	X	X
	X	0	X

Diagram illustrating a 4x4 matrix representing Boolean values. The columns are labeled pre₀ and pre₁. The rows are labeled u₀ and u₁. A blue box highlights the second column (pre₀) and third row (u₀), containing the value 1.

Candidate solution:

$$\psi = \neg \text{pre}_0 \wedge \neg u_0$$

Boolean formula learning (DNF)

		pre ₁	
		0	0
u ₀	X	0	X
	0	1	X
	0	X	X
	X	0	X

pre₀

Candidate solution:

$$\psi = (\neg \text{pre}_0 \wedge \neg u_0) \vee (\text{pre}_1 \wedge u_1)$$

Boolean formula learning (DNF)

		pre ₁	
		0	0
u ₀	X	0	X
	0	X	X
	0	X	X
	X	0	X

$\underbrace{\hspace{10em}}_{\text{pre}_0}$ $\underbrace{\hspace{10em}}_{\text{u}_1}$

Candidate solution:

$$\psi = (\neg \text{pre}_0 \wedge \neg u_0) \vee (\text{pre}_1 \wedge u_1)$$

Boolean formula learning (DNF)

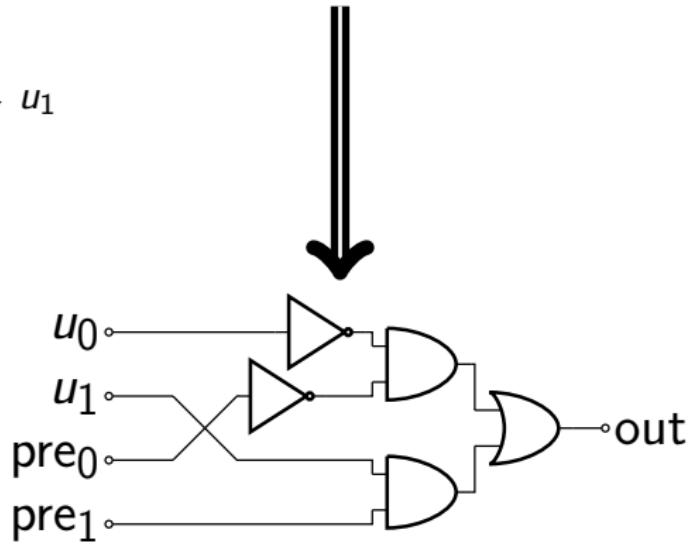
		pre ₁	
		0	0
u ₀	X	0	X
	0	X	X
	0	X	X
	X	0	X

{ } { } u₁

u₁

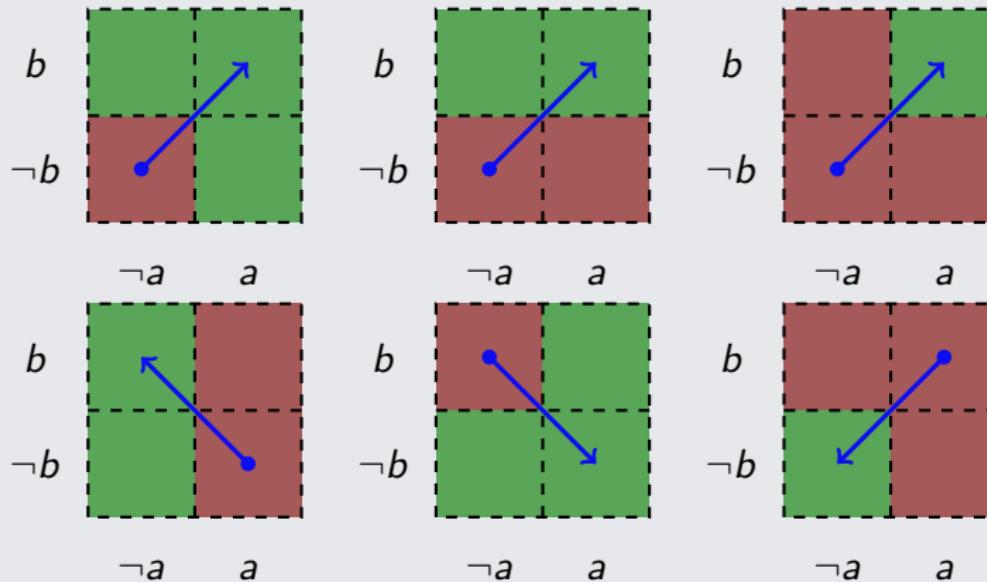
Candidate solution:

$$\psi = (\neg \text{pre}_0 \wedge \neg u_0) \vee (\text{pre}_1 \wedge u_1)$$



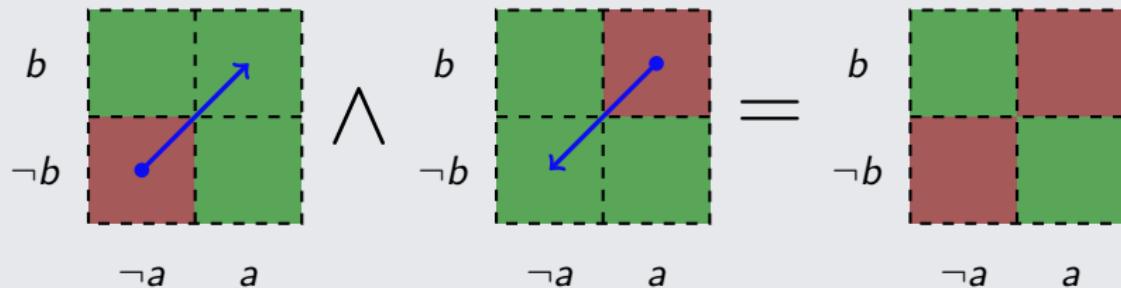
CDNF learning: Bshouty's algorithm

Z-monotone functions



Overlay example: decomposing a XOR into CDNF

Idea



Algorithm

While target function not found:

- Search for **false-positive** z (as in CNF)
If found, add a new z -based DNF
- Search for **false-negative** z (as in DNF)
If found, update all DNFs to accept z

Example: learning a $a \oplus b$

Candidate solution

(true)

Operation

Check for false-positive

Example: learning a XOR b

Candidate solution

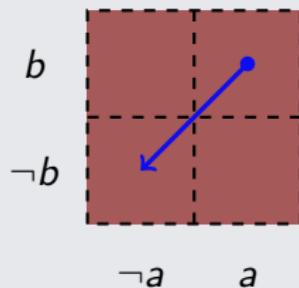
(true)

Operation

Check for false-positive $\rightarrow \mathbf{x}: ab$

Example: learning a XOR b

Candidate solution



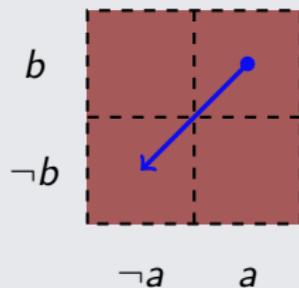
(**false**)

Operation

Check for false-negative

Example: learning a XOR b

Candidate solution



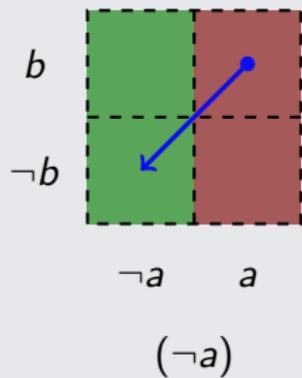
(**false**)

Operation

Check for false-negative $\rightarrow \times: \bar{a}b$

Example: learning a $a \oplus b$

Candidate solution

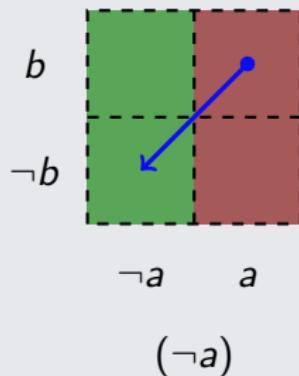


Operation

Check for false-positive

Example: learning a XOR b

Candidate solution

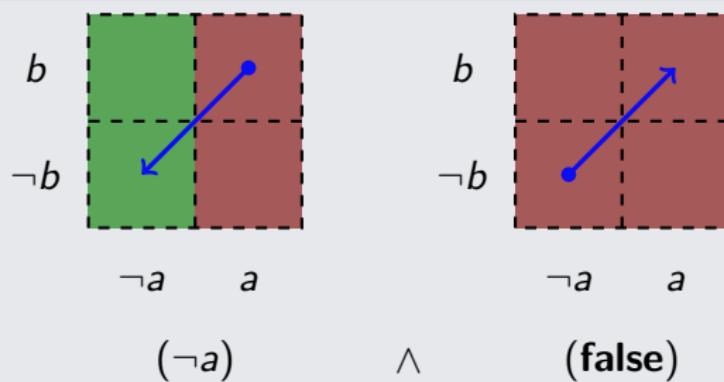


Operation

Check for false-positive $\rightarrow \mathbf{X}: \overline{ab}$

Example: learning a $a \oplus b$

Candidate solution

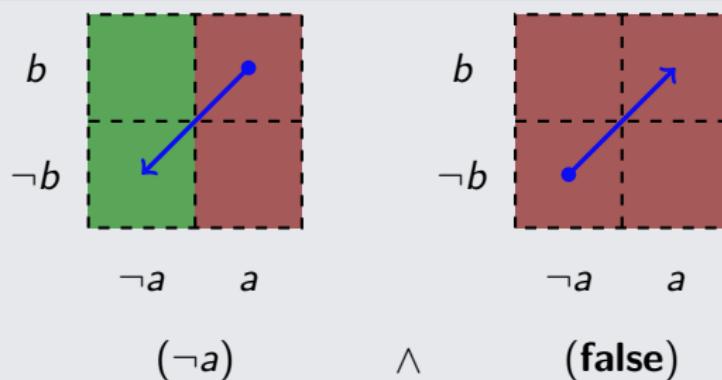


Operation

Check for false-negative

Example: learning a $a \oplus b$

Candidate solution

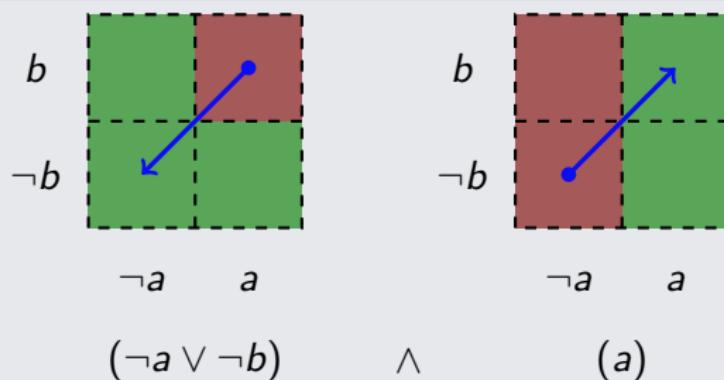


Operation

Check for false-negative $\rightarrow \times: \bar{b}a$

Example: learning a XOR b

Candidate solution

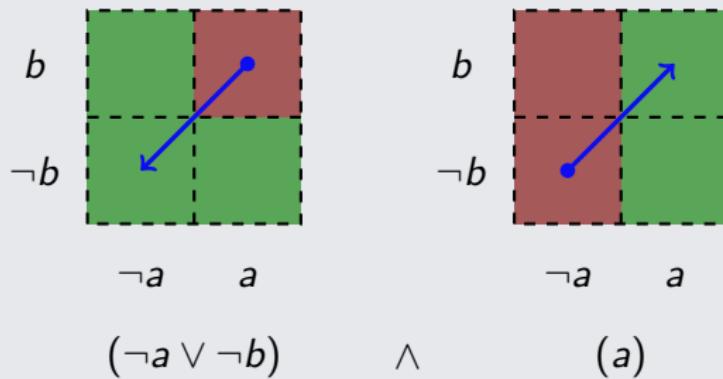


Operation

Check for false-positive

Example: learning a $a \oplus b$

Candidate solution

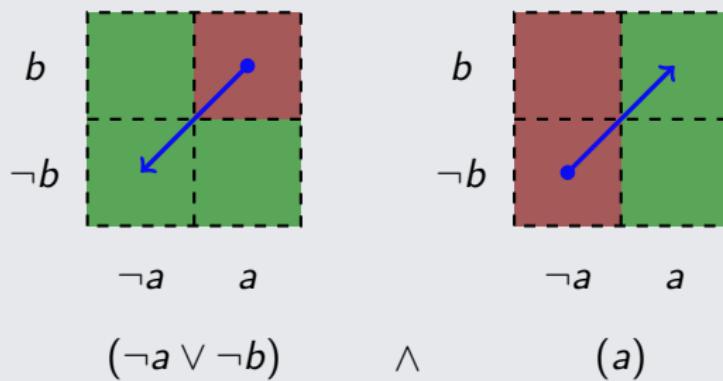


Operation

Check for false-positive → ✓

Example: learning a XOR b

Candidate solution

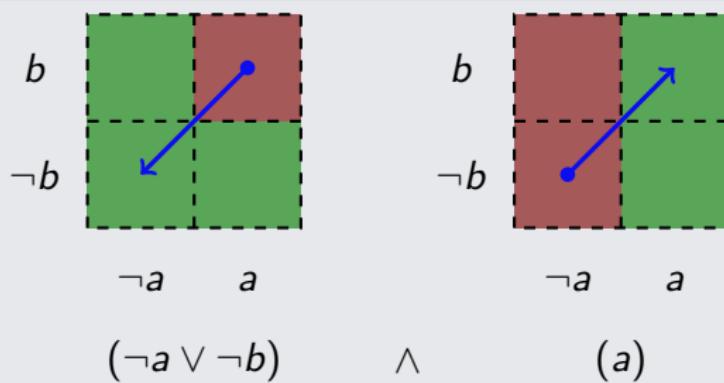


Operation

Check for false-negative

Example: learning a XOR b

Candidate solution

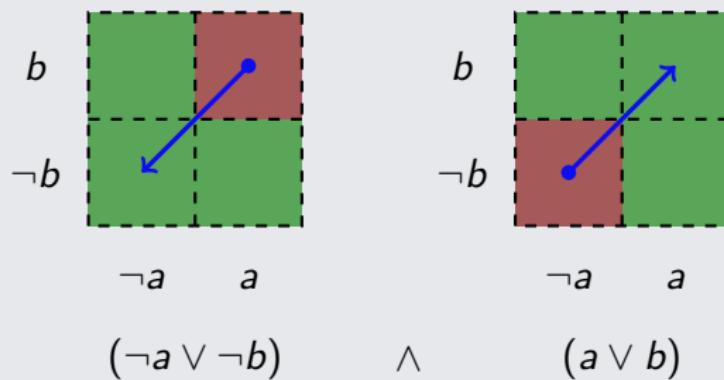


Operation

Check for false-negative $\rightarrow \times: \bar{a}b$

Example: learning a $a \oplus b$

Candidate solution

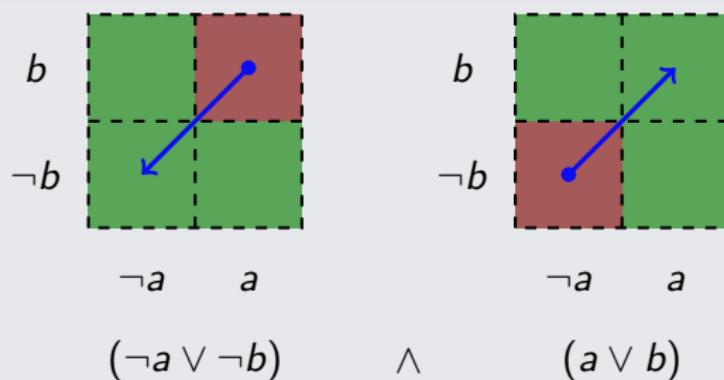


Operation

Check for false-positive

Example: learning a $a \oplus b$

Candidate solution

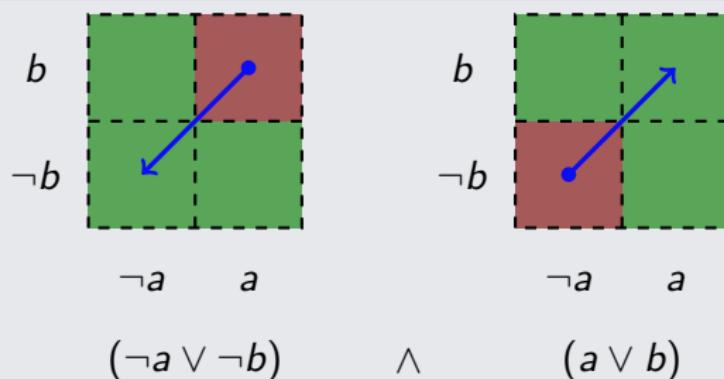


Operation

Check for false-positive → ✓

Example: learning a XOR b

Candidate solution

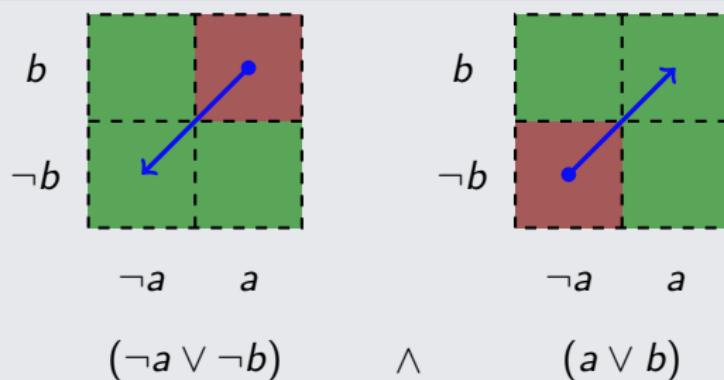


Operation

Check for false-negative

Example: learning a XOR b

Candidate solution

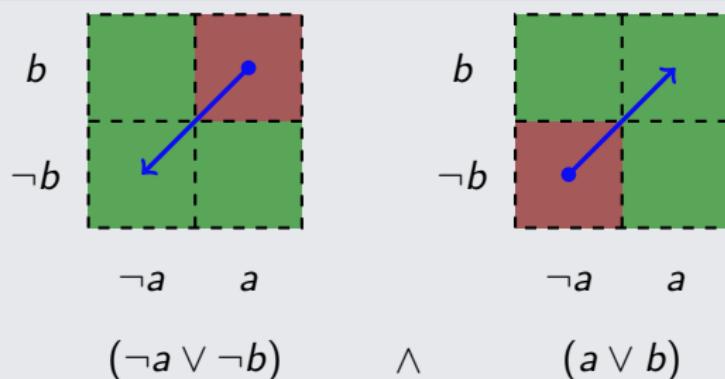


Operation

Check for false-negative → ✓

Example: learning a XOR b

Candidate solution



Operation

No counter-examples? Done!

Experimental Setup

Benchmarks

- RATSY – GR(1) synthesis
 - IBM generalised buffer (Bloem et al., 2007b)
 - AMBA AHB Arbiter (Bloem et al., 2007a,b)
- UNBEAST – bounded synthesis
 - Load balancer (Ehlers, 2010)
 - Various benchmarks by Jobstmann and Bloem (2006)

Implementation

- Uses CUDD as BDD library
- Written in C++
- Imports a general strategy as CUDD BDD (written to a file)
- Postprocessing/verification: ABC and NuSMV (BMC)

Modes of operation

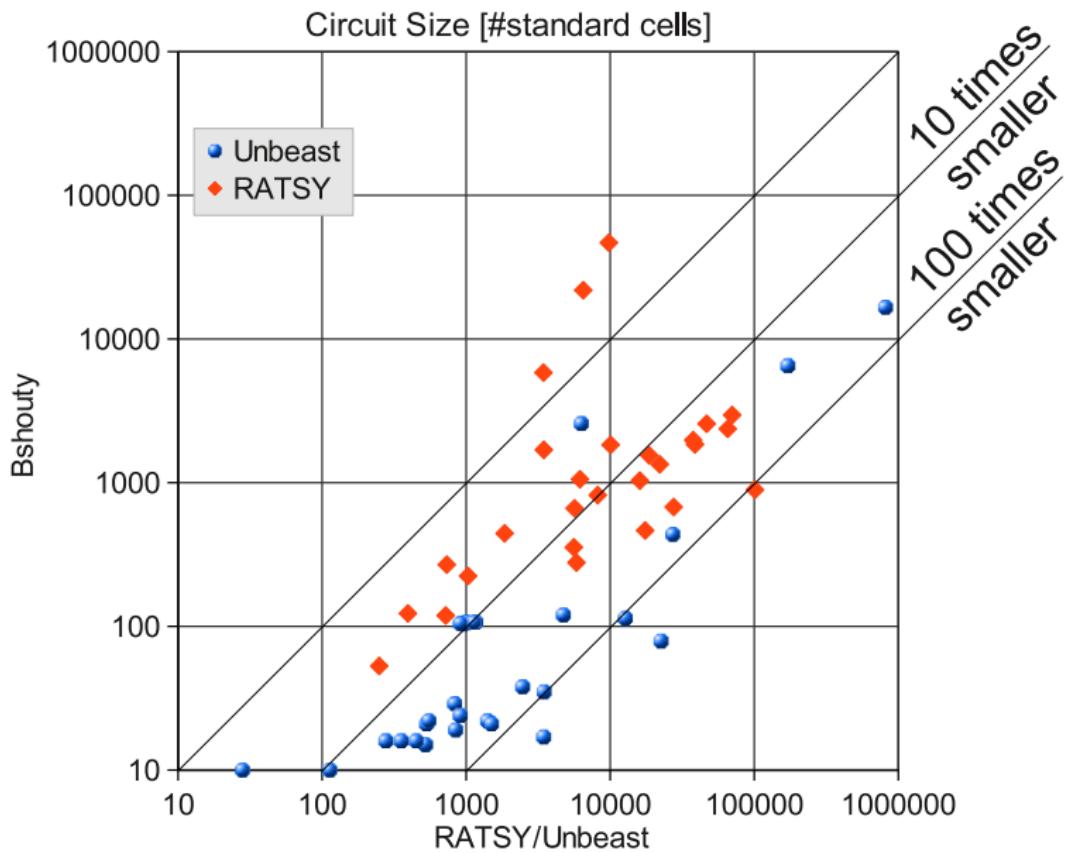
Any combination of ...

- ... DNF/CDNF leaning or their dual case, CNF/DCNF learning
- ... Smart variable order selection on/off

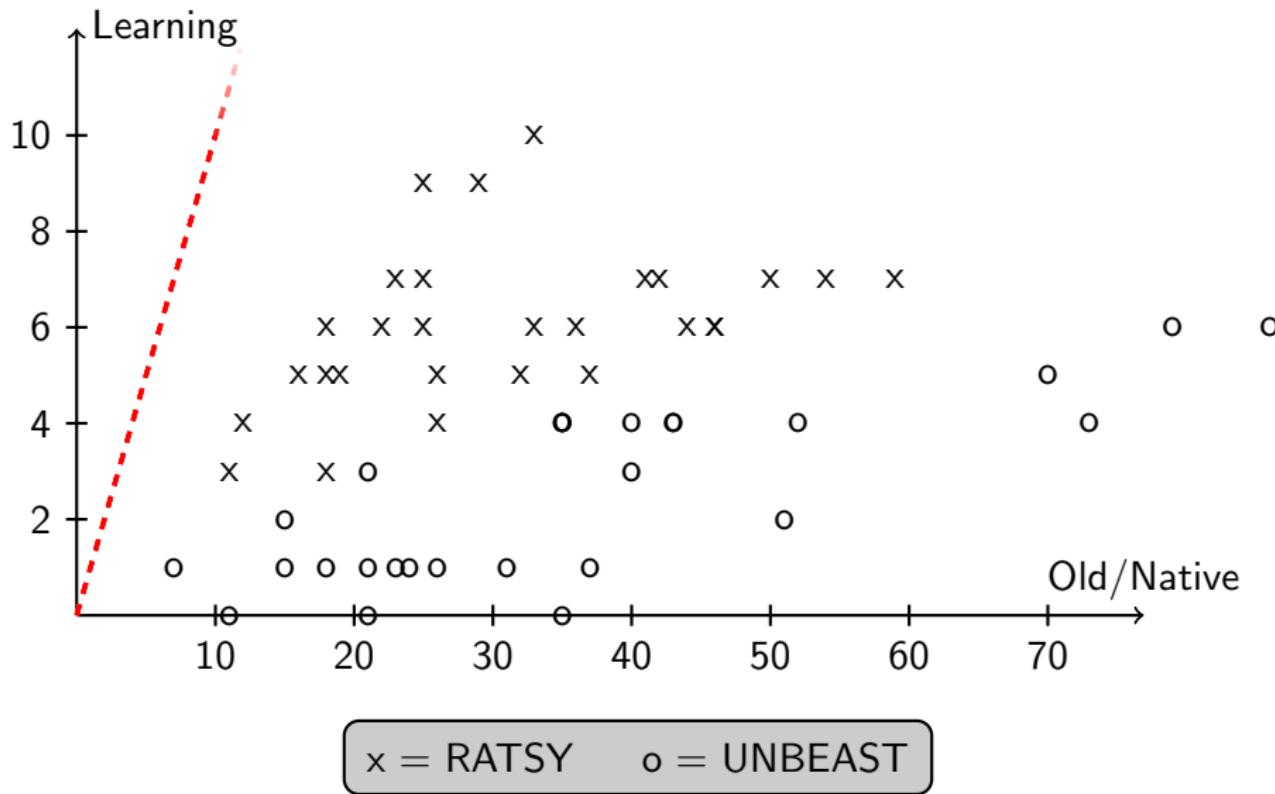
Best modes found

- **CDNF** learning + smart variable order selection **on**
- **DNF** learning + smart variable order selection **on**

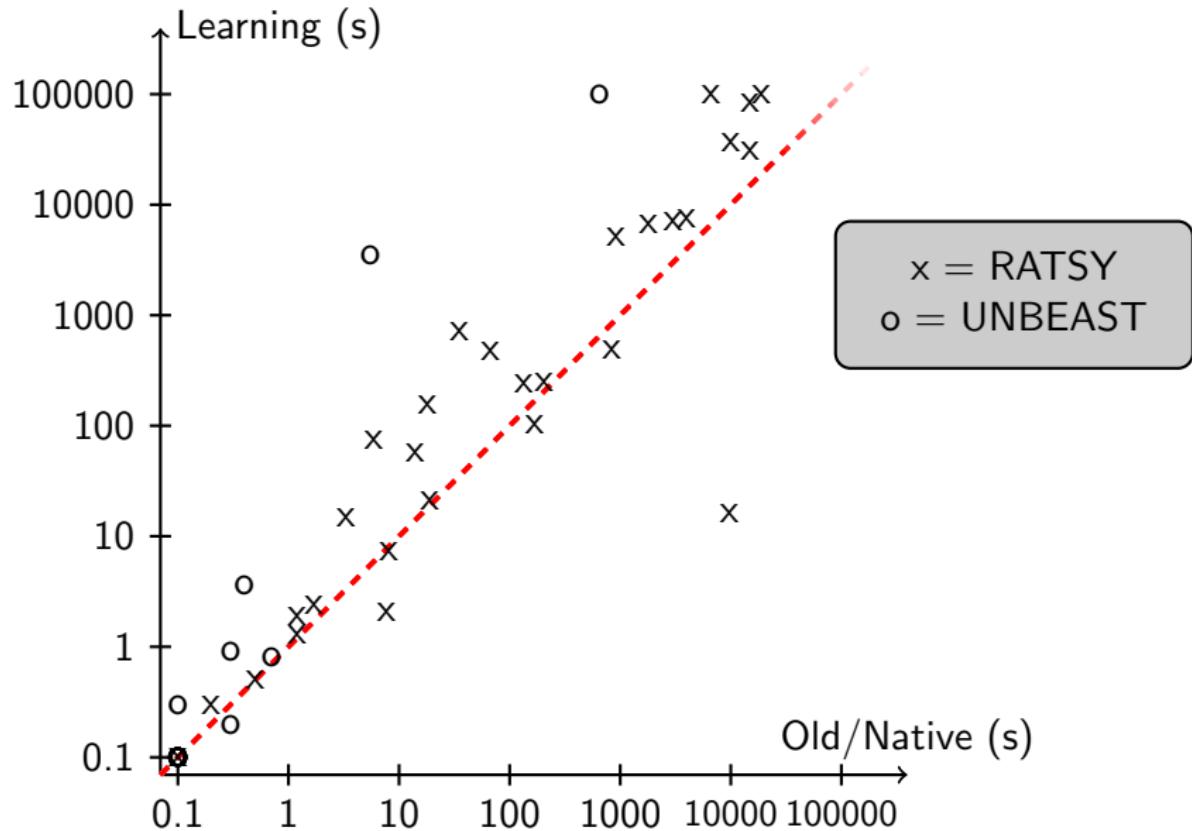
Circuit sizes (CDNF)



Circuit depths (CDNF)



Computation times



Conclusion

Learning-based implementation extraction:

- an *efficient* and *effective* way to compute circuits in reactive synthesis
- *not restricted to BDDs*, but can be used with *any* symbolic data structure
- better suited for the task than all other techniques that we tried



Implementation available (for CuDD)

The URL is in the paper!

References I

- D. Bañeres, J. Cortadella, and M. Kishinevsky. A recursive paradigm to solve Boolean relations. In *Design Automation Conference (DAC'04)*, pages 416–421. ACM, 2004. ISBN 1-58113-828-8.
- R. Bloem, S. J. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Interactive presentation: Automatic hardware synthesis from specifications: a case study. In *Design, Automation and Test in Europe (DATE'07)*, pages 1188–1193, 2007a.
- R. Bloem, S. J. Galler, B. Jobstmann, N. Piterman, A. Pnueli, and M. Weiglhofer. Specify, compile, run: Hardware from PSL. *Electronic Notes in Theoretical Computer Science*, 190(4):3–16, 2007b.
- Rüdiger Ehlers. Symbolic bounded synthesis. In *CAV*, volume 6174 of *LNCS*, pages 365–379. Springer, 2010. ISBN 978-3-642-14294-9. doi: [10.1007/978-3-642-14295-6_33](https://doi.org/10.1007/978-3-642-14295-6_33).
- J. R. Jiang, H. Lin, and W. Hung. Interpolating functions from large Boolean relations. In *International Conference on Computer-Aided Design (ICCAD'09)*, pages 779–784. IEEE, 2009.
- B. Jobstmann and R. Bloem. Optimizations for LTL synthesis. In *Formal Methods in Computer-Aided Design (FMCAD'06)*, pages 117–124. IEEE, 2006. ISBN 0-7695-2707-8.
- J. H. Kukula and T. R. Shiple. Building circuits from relations. In *Computer Aided Verification (CAV'00)*, volume 1855 of *LNCS*, pages 113–123. Springer, 2000. ISBN 3-540-67770-4.